Package 'deep'

October 13, 2022

Type Package

Title A Neural Networks Framework

Version 0.1.0

Author Brian Lee Mayer

Maintainer Brian <bleemayer@gmail.com>

Description Explore neural networks in a layer oriented way, the framework is intended to give the user total control of the internals of a net without much effort. Use classes like PerceptronLayer to create a layer of Percetron neurons, and specify how many you want. The package does all the tricky stuff internally leaving you focused in what you want. I wrote this package during a neural networks course to help me with the problem set.

License GPL-3

Encoding UTF-8

LazyData true

Imports methods

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-12-20 11:50:03 UTC

R topics documented:

deep *deep: A Neural Networks Framework*

Description

The deep package provides classes for layers, types of neurons and the neural network as a whole.

McCullochPitts-class *The McCullochPitts neuron class, that implements the logic of the McCullochPitts neuron model.*

Description

The McCullochPitts neuron class, that implements the logic of the McCullochPitts neuron model.

Arguments

inputs	The actual data to be fed to the nuron, this input's dimentions vary with the chosen weights dimentions.
ins	The list of vectors of inputs to the first layer in the network
outs	The list of vectors of outputs of the last layer in the network
epochs	How many rounds of training to run
tax	This is the learning rate, aka eta
maxErr	A contition to early stop the training process

Value

The computed value using the McCullochPitts model.

Vector of computed values of the same size of the last layer

Fields

ws The matrix of weights that multiply the input vector, it can be a vector, a matrix or an array.

bias The bias value.

Examples

```
# Create a dataset
dataset <- iris
dataset$Petal.Length <- NULL
dataset$Petal.Width <- NULL
dataset <- dataset[dataset$Species != "versicolor",]
dataset$Code <- as.integer(dataset$Species == "virginica")
dataset <- dataset[sample(20),]

# Create the neuron
neuron <- mcCullochPitts(c(1,1), 1)

# Train the neuron, takes a while
neuron$train(dataset[,c(1,2)], dataset[,'Code', drop=FALSE], epochs = 10)

# Check the output
neuron$output(c(1,2))

# See accuracy
dataset$Calc <- sapply(1:nrow(dataset), function(x) {
    as.integer(neuron$output(dataset[x,c(1,2)]))
})
length(which(dataset$Code==dataset$Calc))/nrow(dataset)
```

`McCullochPittsLayer-class`

The McCullochPittsLayer class, that implements a layer of McCullochPitts neurons.

Description

The McCullochPittsLayer class, that implements a layer of McCullochPitts neurons.

Arguments

`input`	The actual data to be fed to the layer, this input's dimentions vary with the chosen `n`.
`ins`	The list of vectors of inputs to the first layer in the network
`outs`	The list of vectors of outputs of the last layer in the network
`epochs`	How many rounds of training to run
`tax`	This is the learning rate, aka eta
`maxErr`	A contition to early stop the training process

Value

The computed value using the McCullochPittsLayer model.

Vector of computed values of the same size of the last layer

Fields

`n` The number of neurons to create in the layer

`dims` A vector of dimensions of the inputs to the layer

`neurons` A list with the internal neurons

NeuralNetwork-class *The main NeuralNetwork class, that holds the layers.*

Description

The main NeuralNetwork class, that holds the layers.

Fields

`eta` The learning tax, representes the size of the weight adjustment between each epoch of training.

`layers` This field is a list of the layers of the network, you can use subsetting to inspect them.

Examples

```
# Create a dataset
dataset <- iris
dataset$Petal.Length <- NULL
dataset$Petal.Width <- NULL
dataset <- dataset[dataset$Species != "versicolor",]
dataset$Code <- as.integer(dataset$Species == "virginica")
dataset <- dataset[sample(20),]

# Create the network
net <- neuralNet(2, perceptronLayer(1))

# Train the network, takes a while
net$train(dataset[,c(1,2), drop=FALSE], dataset[,'Code', drop=FALSE], epochs = 10)

# Check the output
net$compute(c(1,2))

# See accuracy
net$validationScore(dataset[,c(1,2), drop=FALSE], dataset[,'Code', drop=FALSE])
```

`Perceptron-class` *The Perceptron neuron class, that implements the logic of the perceptron model.*

Description

The Perceptron neuron class, that implements the logic of the perceptron model.

Arguments

`inputs`	The actual data to be fed to the neuron, this input's dimentions vary with the chosen weights dimentions.
`ins`	The list of vectors of inputs to the first layer in the network
`outs`	The list of vectors of outputs of the last layer in the network
`epochs`	How many rounds of training to run
`tax`	This is the learning rate, aka eta
`maxErr`	A contition to early stop the training process

Value

The computed value using the Perceptron model.

Vector of computed values of the same size of the last layer

Fields

`ws` The matrix of weights that multiply the input vector, it can be a vector, a matrix or an array.

`bias` The bias value.

Examples

```
# Create a dataset
dataset <- iris
dataset$Petal.Length <- NULL
dataset$Petal.Width <- NULL
dataset <- dataset[dataset$Species != "versicolor",]
dataset$Code <- as.integer(dataset$Species == "virginica")
dataset <- dataset[sample(20),]

# Create the neuron
neuron <- perceptron(c(1,1), 1)

# Train the neuron, takes a while
neuron$train(dataset[,c(1,2), drop=FALSE], dataset[,'Code', drop=FALSE], epochs = 10)

# Check the output
neuron$output(c(1,2))
```

```
# See accuracy
dataset$Calc <- sapply(1:nrow(dataset), function(x) neuron$output(dataset[x,c(1,2)]))
length(which(dataset$Code==dataset$Calc))/nrow(dataset)
```

`PerceptronLayer-class` *The PerceptronLayer class, that implements a layer of Perceptron neurons.*

Description

The PerceptronLayer class, that implements a layer of Perceptron neurons.

Arguments

`input`	The actual data to be fed to the layer, this input's dimentions vary with the chosen `n`.
`ins`	The list of vectors of inputs to the first layer in the network
`outs`	The list of vectors of outputs of the last layer in the network
`epochs`	How many rounds of training to run
`tax`	This is the learning rate, aka eta
`maxErr`	A contition to early stop the training process

Value

The computed value using the Perceptron model.

Vector of computed values of the same size of the last layer

Fields

`n` The number of neurons to create in the layer

`dims` A vector of dimensions of the inputs to the layer

`neurons` A list with the internal neurons

Index